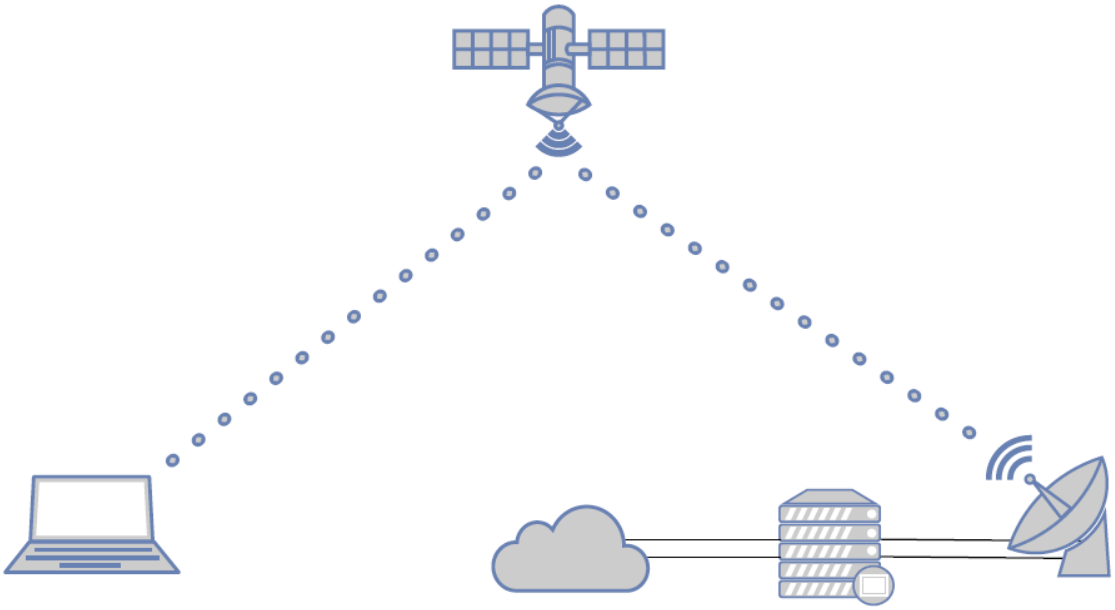
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Reliable and User Friendly Low Bandwidth Web Surfing

Performance and Reliability Improvements of a Proxy Server and Web Browser Prototype

Bachelor of Science Thesis in Computer Science and Engineering

Gustav Pettersson, Loke Simone Damaschke,
Tarik Ala Hadi, Kristoffer Blid Sköldheden

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Bachelor of Science Thesis DATX02-19-14

Reliable and User Friendly Low Bandwidth Web Surfing

Performance and Reliability Improvements of a Proxy Server and Web Browser Prototype

Gustav Pettersson

Loke Simone Damaschke

Tarik Ala Hadi

Kristoffer Blid Sköldheden

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden June 2019

Reliable and User Friendly Low Bandwidth Web Surfing
Performance and Reliability Improvements of a Proxy Server and Web Browser Prototype

Gustav Pettersson, guspette@student.chalmers.se

Loke Simone Damaschke, simoned@student.chalmers.se

Tarik Ala Hadi, tarika@student.chalmers.se

Kristoffer Blid Sköldheden, guskrissk@student.gu.se

Supervisor: Dag Wedelin
Examiner: Carl-Johan Seger

Bachelor of Science Thesis DATX02-19-14
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The illustration pictures the connection between the computer and the Internet, using a satellite connection and a proxy server. (T. Andersson, L. Blomkvist, A. Hast, F. Karlsson, J. Lindström, and T. Sundell, Very Low Bandwidth (Marine) Web Surfing A Fault-Tolerant Content Streaming Web Browsing Solution, 2018)

Gothenburg, Sweden 2019

Reliable and User Friendly Low Bandwidth Web Surfing
Performance and Reliability Improvements of a Proxy Server and Web Browser Prototype

Gustav Pettersson, Loke Simone Damaschke,
Tarik Ala Hadi, Kristoffer Blid Sköldheden

Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

In remote locations, internet access can be enabled using satellite connections, such as the Iridium Satellite Network. The Iridium GO! device provides a bandwidth of 2.4 kbit/s. On such a low bandwidth it would take hours to download most modern websites.

In 2018, a prototype which enables general web browsing on a low bandwidth was developed at Charmers. The prototype acts as a proof of concept for the use of a performance enhancing proxy server to extract data in order to reduce content size and achieve viable loading times.

The 2019 prototype replaces components of the 2018 iteration, providing a faster and more reliable service. In particular, no excess data is being sent and the delay caused by compression and data extraction is notably shorter, resulting in a 66% reduction on average to the time between sending a request and the moment the first content arrives to the client.

Keywords: satellite Internet, low bandwidth web browsing, Iridium Satellite Network, performance enhancing proxy, content extraction.

Reliable and User Friendly Low Bandwidth Web Surfing
Performance and Reliability Improvements of a Proxy Server and Web Browser Prototype

Gustav Pettersson, Loke Simone Damaschke,
Tarik Ala Hadi, Kristoffer Blid Sköldheden

Department of Computer Science and Engineering
Chalmers University of Technology

Sammandrag

På avlägsna platser så kan tillgång till internet ges av satellitanslutningar, som till exempel Iridium Satelite Network. Produkten Iridium GO! ger tillgång till en uppkoppling på 2.4 kbit/s. På en sådan långsam uppkoppling skulle de ta fleratal timmar att hämta de flesta moderna hemsidor.

2018 utvecklades en prototyp på Chalmers, som ett realistiskt alternativ för att surfa på denna långsamma uppkopplingen. Prototypen agerar som ett konceptbevis som med hjälp utav en proxy-server extraherar nödvändig data för att reducera dess storlek, för att sedan skicka vidare denne till användaren, som resulterar i rimliga laddningstider.

2019 års prototyp har bytt ut komponenter från 2018 års variant för att förbättra hastigheterna och göra tjänsten mer pålitlig. Bland annat så skickas ingen onödig data, och fördröjningen som orsakas av komprimering och extrahering har reducerats. Detta resulterar i ca 66 % reduktion i tid från när förfrågan görs av användaren, till att första datan anländer hos klienten.

Nyckelord: satellit-internet, låg bandbredd, webbsurfande, Iridium Satellite Network, presationsförbättrande proxy, extrahering av innehåll.

Acknowledgements

We would like to thank Dag Wedelin for his guidance as supervisor for this project. He was of tremendous help in directing the focus of the project and offered unique insight with his experience using the 2018 prototype in the field. He also provided his personal Iridium GO! device for testing.

We would also like to thank everyone who offered feedback and input throughout the project, both for the prototype and the thesis.

Contents

Glossary

DOM Short for "Document Object Model". A DOM represents the structure of a HTML or XML document as a tree.

HTML Short for "HyperText Markup Language". A markup language used to structure web pages.

Network protocol A network protocol specifies how communication between software is conducted via a network, defining accepted values.

Packet A sequence of bits which is sent as one unit between devices within a network.

Packet Header The part of a packet which contains control information for addressing.

Payload The part of a packet which contains the transmitted information.

PEP Short for "Performance Enhancing Proxy Server", a server which acts as intermediary between two end systems in a network and performs operations to enhance system performance.

TCP Short for "Transmission Control Protocol", a communications protocol running on top of IP which provides in-order delivery.

UDP Short for "User Datagram Protocol", a communications protocol running on top of IP which is loss-tolerant.

1

Introduction

While high-speed internet access is prevalent in densely populated areas, the same does not hold true for locations away from cities and towns. Regular internet connections are rooted in place or dependent on range from a cellular tower [1]. When sailing or hiking to remote locations, internet access can be enabled through satellite communication.

There are different satellite connection services available. Enterprise grade satellite connections offer high bandwidths, but are costly. Using a TracPhone, a 10/3 Mb/s connection can cost 0.5 $ per Mb [2]. More affordable variants are marketed to individual hobbyists, such as the Iridium GO! device, which provides a bandwidth of 2.4 kbit/s [3] at up to 0.89 $ per minute of connection [4]. However, a modern web page, such as the English Wikipedia page about Sweden, would take about an hour to download using the 2.4 kbit/s connection.

While there are satellite connections with sufficient bandwidth for web browsing, it is a question of price. This begs the question if general web browsing can be enabled on an affordable low bandwidth connection. The goal of this project is to develop a robust and reliable low bandwidth web browsing solution which can be utilised by a user without specialised knowledge.

There are several custom applications for specific satellite web browsing needs, but they either limit the user to specific sites or information, such as weather reports or Facebook and Twitter use [5], or do not reduce the content size enough to work reliably with a bandwidth as low as that of the Iridium Satellite Network. For example, the general satellite web browsing solution XWeb only reduces web page content by a factor of 3-5 [6].

In order to stream web content on a low bandwidth connection, an aggressive reduction in content needs to be ensured, while preserving all the vital textual informa-

tion. Furthermore, saving data can reduce costs for the user. On a regular internet connection, users often have unlimited or very generous data plans, but this is not always the case on a satellite connection.

Filtering the vital textual content from the non-essential is a complex problem. In the infancy of the internet, websites consisted mainly of text and links, but modern websites contain a lot more information, such as complex layouts, images, and video content. Each type of media and each design element may have several different HTML tags which identify it, and each such variant has to be taken into account when filtering the content.

1.1 Purpose and Scope

The purpose of this project is to enable general web browsing in remote locations in a cost-effective and timely manner. While the project focuses on creating a robust and reliable general web browsing solution using the Iridium GO! device, the work may have broader implications for data extraction and transfer to remote locations via other networks and technologies.

In 2018, a Bachelor of Science thesis was published at Chalmers on the topic of low bandwidth web browsing. The thesis describes a prototype which employs a proxy server to filter web content before streaming the filtered data to a client via the Iridium Satellite Network [3]. The prototype also includes a set of custom, minimal protocols which further reduce the amount of information sent via satellite. This solution appears to be unique and other research on enabling general web browsing using the 2.4 kbit/s Iridium Satellite Network connection was not found.

While not constituting a viable product, the 2018 prototype is a proof of concept which forms the basis for continued development to enable general low bandwidth web browsing. During this project, the content extraction of the 2018 prototype was refined and the implementation of the server and client applications altered to improve robustness and usability.

The most important issues to tackle in order to create a reliable service were bug fixes, error handling, and speed improvements. Speed improvements necessitate optimisation, involving problem analysis, development efforts, and testing to confirm whether the new solution really constitutes an improvement. This optimisation process was the main focus of the project.

Multimedia processing was left for later stages of development which may occur after the conclusion of the current project. Any changes to hardware or improvements to the Iridium Satellite Network itself were also outside the scope of the project.

The user interface is undeniably important in any application which is aimed at an end user. Due to time constraints, the user interface was not extensively tested or fleshed out during the project, but it has been taken into account when considering the overall design of the system in order to ease future development which may occur after the conclusion of the current project.

One part of the user interface is the graphical representation, the GUI. Development of a more suitable GUI is not within the scope since the 2018 design suffices for the needs of the project. The GUI provided by the 2018 prototype was used to test the application and only altered insofar it benefited the code base of the underlying system.

2

The 2018 Prototype

In order to understand the changes and improvements made during this project, it is vital to first gain an understanding of the 2018 prototype which has been developed and described in a previous Bachelor thesis. This 2018 prototype will be introduced in this Chapter; how it operates, why, and what aspects need improving. The theory presented is partially extracted from the 2018 thesis and partially the result of further investigation and testing of the 2018 prototype. Underlying theory and implementation specifics which are not relevant to the current project are not presented here. For further information on the 2018 prototype, please refer to the 2018 thesis [3].

2.1 An Overview of the System

The system consists of three main parts, which can be seen in Figure 2.1. The first part is a custom client application which the end user utilises. The client is graphically represented as a simplified web browser. The second part is a set of custom protocols used on the low bandwidth connection between the client and proxy server. The third part of the system is a proxy server which also has a high bandwidth connection via which it can access the Internet.

A proxy server is a server acting as intermediary between a client and a server. One category of proxy servers is performance enhancing proxy servers (PEP). A PEP can help improve Internet protocol performance when standard protocols are unreliable or slow due to the nature of the underlying technology used for at least a part of the network [7][8]. The PEP used in the 2018 prototype operates on several layers, changing the content being transmitted, as well as how it is transmitted.

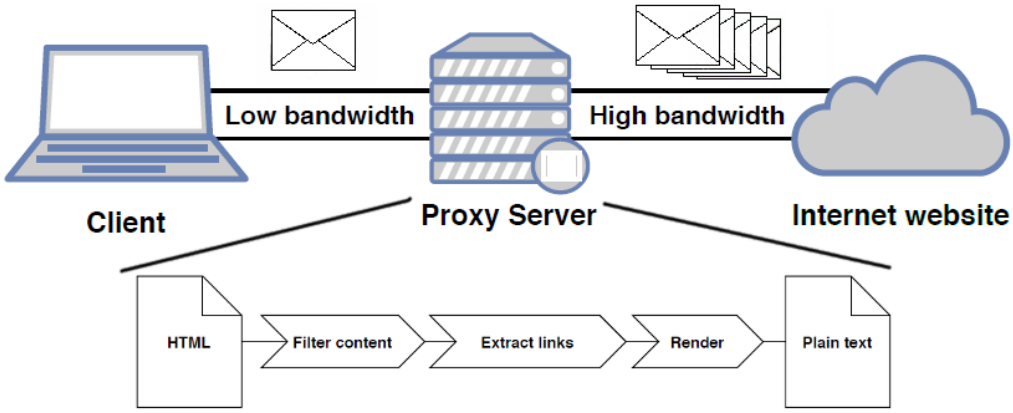


Figure 2.1: Performance enhancing proxy server (edited with permission) [3]. The figure shows how the different parts of the system are related to another. The client and proxy server are connected via a low bandwidth connection where only little information is sent. The proxy server is connected to the internet via a high bandwidth connection where a lot of information is sent. On the proxy server, a process converts HTML documents into plain text.

When the client requests a web page, the request is sent to the proxy server via the low bandwidth connection. The server uses its high bandwidth Internet connection to download the requested page. The server then extracts the vital textual information before sending the extracted content to the client via the low bandwidth connection, using custom low-overhead protocols. This system design greatly reduces the amount of information sent via the low bandwidth connection, thus enhancing performance and enabling general text based web browsing in a timely fashion. This filtering of information also saves data, which potentially reduces the cost of transmission.

The custom protocols used on the low bandwidth connection are not only minimal but also enable out-of-order packet delivery and bit error handling, which reduces the need to resend packets if errors occur. Resending is undesirable since it increases the amount of information that needs to be sent on the low bandwidth connection.

In general, when transmitting data over a network, bit-errors can occur. The data that arrives to the client can thus differ from the data sent from the server, in the worst case making it useless or even harmful to the client. While data sent via the Iridium GO! device does not appear to have any significant bit-error issue, the 2018 prototype is designed to potentially work with other, more bit-error prone services [3]. This risk of data corruption during transmission is thus actively addressed in the 2018 prototype and remains an integral part to the system design.

Handling bit-errors necessitates that every part of the communication takes the pos-

sibility of an error into account, including encryption, compression, and verification of data. How these issues are addressed in the 2018 prototype is detailed in the relevant sections below.

2.2 The Iridium Satellite Network

There are different technologies which can enable internet access in remote locations. Among communications satellite solutions there are two categories of note. The first is GEO (geosynchronous) satellites. GEO satellites orbit the earth in time with the earth's rotation, which enables them to always cover the same area on the surface of the earth. Due to their altitude, GEO communications satellites suffer high latency [9].

The second category is LEO, (low earth orbit) satellites. LEO satellites are, as the name suggests, in low orbit, meaning they are close to the surface and delays are short, making them suitable for communications solutions. LEO satellites are however not geosynchronous and one satellite cannot cover the same area on the surface of the earth for very long. The Iridium Satellite Network handles this complication by employing at least 66 active satellites which orbit the earth in formation, as shown in Figure 2.2. Using this formation, the Iridium Satellite Network enables worldwide telecommunications.

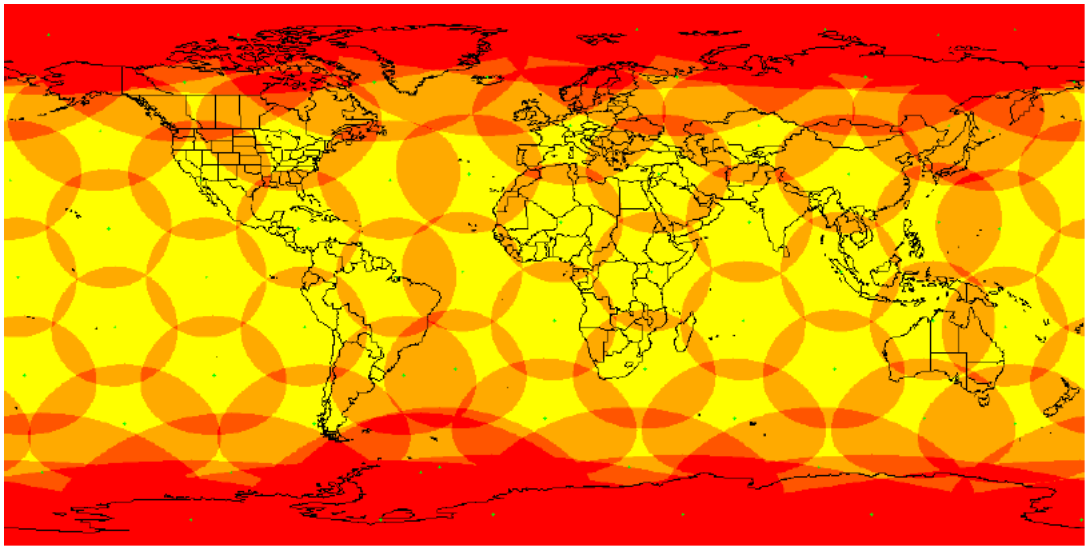


Figure 2.2: Iridium Satellite Network coverage (one image of a gif, not altered) [10]. Each red circle represents the coverage of one satellite. Together, 66 circles cover the entire map.

The original purpose of the Iridium Satellite Network was enabling voice and plain text transmission, such as fax or traditional text files [11]. The connection thus has a bandwidth of only 2.4 kbit/s [3]. The Network forms the low bandwidth connection for the prototype, meaning the communication between server and client is sent via the Iridium Satellite Network. An Iridium GO! device is used to connect to the satellite network.

The next generation of Iridium satellites is already in orbit [12]. The Iridium NEXT satellites promise higher bandwidth and inter satellite communications [13].

During this project, a test was performed to confirm the bandwidth in use. A data stream was sent with UDP (User Datagram Protocol), using the program iPerf3[14], via the Iridium GO! device. The transmission time was measured and the test confirmed that the average speed of the service is indeed 2.4 kbit/s.

2.3 Data Extraction and Rendering

When retrieving a web page, the proxy server notifies the content server from which it is requesting the page that it is a mobile phone in order to get the potentially smaller mobile version of the page if such a page exists. It has been found, however, that the difference between requesting a mobile page versus a regular page is very small once the content extraction is done [3], meaning the act of notifying the content server and acting as a mobile phone does not have any significant impact on the final content size. Acting as a mobile phone may cause a difference in content, though, since some web pages offer drastically differing information and services to mobile user. Nevertheless, changing this is not a priority and the mobile version is in use.

After retrieving the web page, the server performs three operations; filtering, link extraction, and rendering as seen in Figure 2.3. Once the plain text is ready, the content is sent to the client.

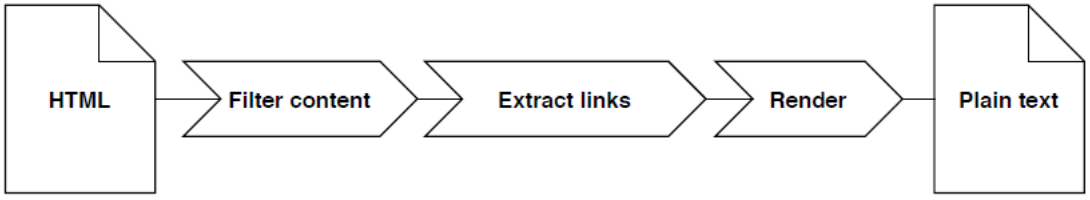


Figure 2.3: Extraction flow chart [3]. The figure shows the process of extracting text from a web page running on the server. The input is a HTML document and the output is a plain text document.

The first step of content extraction is filtering. The HTML elements present in the document are matched to known HTML elements and entirely removed if they are used for styling, scripts, forms, or multimedia, leaving only the textual content (encoded with UTF-8) intact. This greatly reduces the size of the web page.

Any links on the web page are replaced with an ID. The full URL of the link is kept by the server while the client only receives an ID, as well as the starting and ending position of the clickable text. While HTML uses <a> and to identify the start and end of a link, the replacement link ID uses the hexadecimal numbers Oxfe and Oxff. Oxfe and Oxff were chosen because they are not used in UTF-8 and thus do not create any conflict [3].

Since the ID is much smaller than a URL, this process reduces the amount of data sent to the client further. The link ID can be used by the client to request the web page related to it, and the ID will be matched to it's respective link by the server upon such a request. Since web pages can contain thousands of links, this process can take considerable time and the list of links the server needs to keep can grow rapidly. The server can be utilised by several clients simultaneously, amplifying this issue.

Once the textual information has been extracted and links have been replaced, the text-based web browser ELinks is used to render the content, which creates a minimal but easily readable plain text output [15]. When rendering is completed, the content can be sent to the client.

2.4 Server-Client Communication

The two standard protocols used for web browsing on a regular internet connection are User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) [1]. TCP performs poorly with satellites due to the long delays inherent in the technology, resulting in slow throughput of data [9]. In contrast to TCP, UDP allows for out-of-order delivery. Furthermore, automatic resending of corrupt packets can be and is for the prototype disabled using UDP. Finally, UDP has a comparatively small header (8 bytes versus TCP's 20 bytes). Since speed of delivery and data reduction is more important than reliant delivery of packets on slow connections, the custom protocols run on top of UDP [3].

For a detailed account of the custom protocols, please refer to the 2018 thesis [3].

The prototype also has a simple encryption. As is, the encryption does not protect against active attacks, where a bad actor changes the data during transmission. Since it is undesirable to resend packets which contain bit errors on a low bandwidth connection, simple message authentication can not be implemented. Message authentication would mark packets which contain bit errors as invalid instead of accepting them.

2.5 Compression Scheme

The compression algorithm chosen for the prototype is Byte-Pair Encoding (BPE). BPE replaces commonly occurring pairs of bytes with unused byte codes in an iterative process [16]. The text sent to the client is encoded with UTF-8. 102 of 104 UTF-8 codes are not in use and are utilised by the compression algorithm [3].

Usually, BPE would require transmission of a dictionary defining the unused bytes and their corresponding byte-pairs. Sending a dictionary on the low bandwidth connection is infeasible both because it means sending more data and because the custom protocols allow for bit errors as described in Chapter 2.4. If a bit error occurs during the transmission of the dictionary, the dictionary may be corrupted. The dictionary is instead precompiled for a representative website. The precompiled dictionary is used by the server to compress and by the client to decompress.

The main advantages of BPE are that each packet can be decompressed as soon as it arrives and that it is only partially vulnerable to bit errors. If a bit error occurs, the

byte pairs compressed into the corrupted byte are affected, but not the rest of the text. In this way BPE ensures that bit errors do not escalate during decompression.

On average, the compression algorithm of the 2018 prototype achieves a 72% compression [3]. The use of a precompiled dictionary reduces the effectiveness of the BPE compression, but sending a dictionary via the low bandwidth connection is, as mentioned, not feasible. In order to improve the performance, two separate dictionaries are precompiled, one for Swedish and one for English. The text is compressed using both dictionaries, and the smaller version of the text is subsequently sent [3]. While improving the compression rate, this double compression approximately doubles the run-time of the compression.

2.6 UI and GUI Design

The user interface (UI) is the overall design of the user-system interaction. One part of it is the graphical user interface (GUI). While they are closely related, UI and GUI are not the same thing. While the GUI includes colours, shapes, placement on the screen, and other graphical details, the UI also includes what input is required from the user and what responses the system provides on a more general level. While GUI design is not within the scope of the project, the UI strongly influences system design and is thus taken into account. The UI and GUI of the 2018 prototype are briefly described below.

Figure 2.4 shows the client application after the web page example.com has been retrieved.

At the bottom of the screen, the status bar displays the connection status of the client application to the server, connected or disconnected, as well as the current loading status. The feature in the middle is blank when the application is started. When loading a web page, the feature contains images which provide the user with information about the currently active process. When a web page has been retrieved, the feature displays the filtered content of said web page. At the top of the screen, are three navigation buttons and an address field. Above them is the menu, which only provides links to Electron help pages, a full screen toggle, and an option to close or open the application.

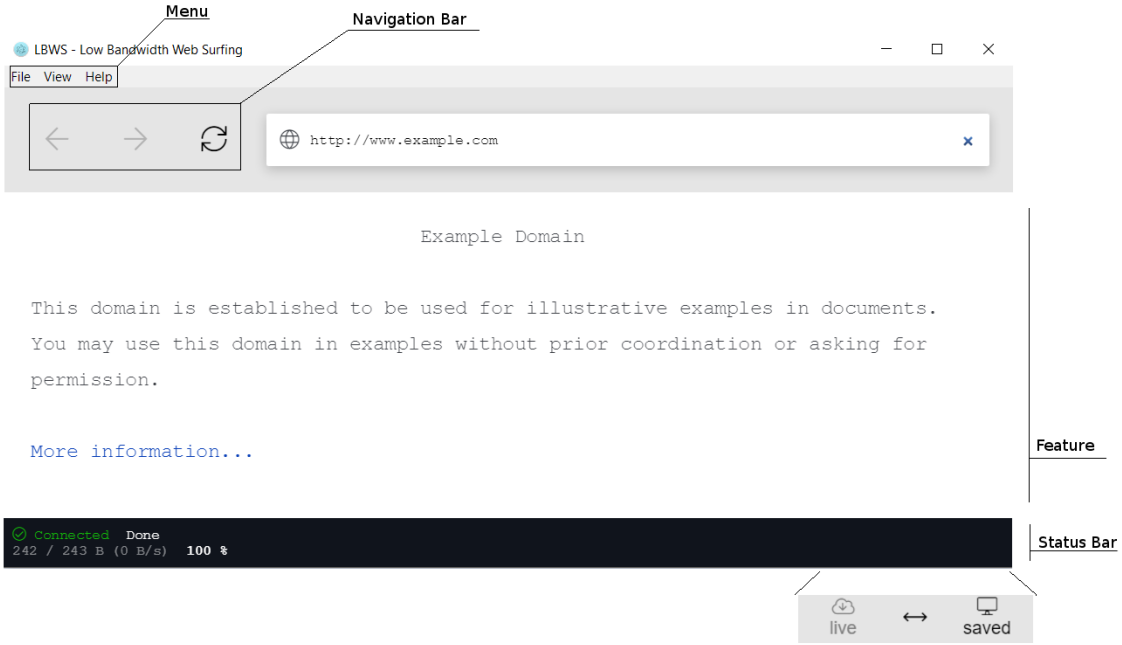


Figure 2.4: Client application running on Windows and showing example.com. The menu and navigation bar are at the top along with the address field. The feature is in the middle and the status bar, which also displays a cache toggle when a cached page is available, is at the bottom.

2.6.1 Search Functionality via Google

When making a request, the user can either provide the full address to a specific website or can provide a search term. Whenever an input has a blank space or is missing a dot, it is interpreted as a search request rather than as a website address. Any search request is converted into an address request by concatenation to "https://www.google.com/search?q=". For example, the input "car" will result in the request "https://www.google.com/search?q=car".

It is today common to find search functionality in the address field of a web browser. When entering a search term, the browser picks a default search engine and performs the search directly. In the 2018 prototype GUI, the same principle holds true. Both website and search requests are entered into the address field. This solution, however, does limit the user to simple searches, unless the user knows how to form an advanced search query directly via the address by adding quotation marks or logic operators.

2.6.2 Browsing History and Navigation

When a user requests a website, the URL is saved to an external history file. When there is a website saved in the browsing history before the one currently being displayed, navigation to the previously visited page is possible. When there is a website saved in the browsing history after the one currently being displayed, navigation to the following visited page is possible. Upon navigating within the browsing history, the respective URL is retrieved from the history file.

The two leftmost navigation buttons are used to perform this navigation of the browsing history, backwards and forwards. The third navigation button cancels a request, stopping the process of retrieving a website from the server. When a website is retrieved, the cancel button instead acts as a reload button, which requests the same page anew. This follows the same pattern as most modern web browsers, making it intuitive for the user. Due to habituation, it may even appear strange if this were not the case [17].

2.6.3 Keeping visited Pages in Cache and the Cache Toggle

One way to reduce the amount of information sent via the low bandwidth connection is to not send any content at all. If a web page is cached in the client, it can be displayed anew without retrieving the page from the server.

In the 2018 prototype, a simple cache is implemented which saves each visited website to an external file. When a user returns to a previously visited page by navigating the browsing history, the page is loaded from the cache file. However, a request is also sent to the server in order provide the user with an updated version of the page as soon as possible. This automatic request means data is being sent over the low-bandwidth connection in a way that is non-essential and not strictly self-evident.

The 2018 GUI provides a toggle which enables the user to switch between the cached page and the page which is being downloaded in parallel. The user can switch back and forth between these freely and is informed which version of the page is currently displayed by the visuals of the toggle button at the bottom of the screen. When the cached page is being displayed, there is also a message in a banner at the top of the feature, informing the user about the cached state and how to switch to the updated version. This toggle is not a standard GUI component of a web browser and thus runs the risk of being unintuitive for the user [17].

3

Development of a 2019 Prototype

As described, the 2018 prototype is a proof of concept for the use of a PEP to extract data from websites in order to enable general web browsing on a low-bandwidth, such as a satellite internet connection. Specifically, the Iridium GO! device is used for the development of this system. The 2018 prototype is not a viable product that can be given to an end user, especially not a non-expert end user. Continued development is needed in order to create a viable product. During this project a new 2019 prototype was developed based on the previous 2018 prototype. How this development was performed is described here.

The continued development effort consisted of six main processes or aspects; initial testing, requirements specification, simplification, performance improvements, UI considerations, and documentation. Some of these aspects are processes performed in a specific order, while others are ongoing efforts or considerations that influence the development. Each aspect is discussed in detail below.

3.1 Initial Testing and Fact-finding

In order to establish a baseline from which work can begin, the project included a lengthy initial testing and fact-finding process. Literature relevant to the project, including the 2018 thesis and many of its sources, were studied. The code base was received from the original authors and both investigated by reading the code and by running tests to determine how well the 2018 prototype performs.

In quantitative tests, two metrics were studied; time to first content and loading time. While absolute time matters for the experience of the user, different machines with differing hardware will perform at vastly different speeds and improvements in these metrics thus have to be studied as relative values on the same machine. When

test data was collected, evaluated, and presented in this thesis, comparisons were made accordingly, for each test using values produced running different versions of the code on the same machine.

Time to first content is the time form the moment the request is sent until the first content arrives to the client. Time to first content was the most critical measurement guiding the development of the software since it is the primary point of frustration when using a low-bandwidth web browser. Waiting an hour, as in the Wikipedia example given previously, to see if the web page is loaded or not is not a user friendly experience.

Furthermore, users can be assumed to be accustomed to responsive systems and web browsers which are able to display websites almost instantly. When it takes a long time to load content, the user is likely to experience the service as inferior in quality, and even as unreliable. When interacting with a web page, a delay of over 10 seconds is experienced as a long time by many users, especially those who spend a lot of time browsing the internet [18]. This delay and user experience data was generalised, given that no research could be found on the specific circumstances of the current project. The relevant study concerns web pages and while the project was about a system for the loading of web pages. While imperfect, due to differing demographics and circumstances, this generalisation appears fairly reasonable.

While users of the system will be aware that they are on a satellite connection with a low bandwidth, and can thus be expected to have more patience than on a standard connection, frustration can accumulate, meaning users are likely to become less tolerant to delays as they spend more time using a slow web page [18]. Again, it was assumed that the same holds true for the prototype. Delays should thus be as short as possible to avoid frustration buildup.

Similarly, loading time is of importance. Loading time is the average time it takes to load web content. The loading time was measured from the time of first content until the entire website is loaded, and then divided by the size of the website. Bigger websites will always take longer to load, but if the loading time is small, the user will see new content continuously, making the system appear responsive.

3.2 Requirements Specification

The goal of the project was, as stated, to develop a robust and reliable product which an end user can utilise to browse the web over a low bandwidth connection. Robustness was achieved by addressing bugs within the system and handling errors

which may occur. The user's experience of reliability depends greatly on time to first content as well as loading time, as explained above.

The goal of the development effort was to create a minimum viable product (MVP), a product which can be used by a non-expert user as is. The MVP needs to fulfil certain requirements. The requirements were separated into functional and non-functional. Functional requirements describe the functionality that the system must provide to the user. Non-functional requirements describe the quality constraints. The non-functional requirements were further separated into execution and evolution qualities. Execution qualities are apparent to the user, such as usability and responsiveness, while evolution qualities concern the underlying architecture, how well it scales, and can be tested and adapted in the future.

Functional requirements:

- Upon receiving a website address as input, the system must return the textual information of the appropriate website.

- Upon receiving a search term as input, the system must return the textual results of the appropriate search, using any search engine.

Execution qualities:

- Responsiveness: The time to first content shall not exceed 10 seconds, and should be as short as possible.

- Learnability: The web browser should conform to the layout that users are accustomed to and expect from a standard web browser.

Evolution qualities:

- Extensibility: The system design shall have clearly separated and individually replaceable components. The code base should be easy to read and alter.

- Scalability: The system design should not constrain the amount of users or websites.

While more, and more detailed requirements could be defined, these six were deemed detailed and extensive enough for the scope of the current project. The functional

requirements are mostly met by the 2018 prototype, with the exception of some error handling and bugs.

The time to first content of the 2018 prototype is already below 10 seconds when requesting small to medium web pages, but for large web pages it can take over 16 seconds. Time to first content could certainly be improved. In order to improve on time to first content, the bottleneck had to be identified and the responsible part of the process subsequently optimised. The bottleneck was identified by implementing timers for the various processes running in the system and comparing their run-time.

At least one potential learnability issue is the use of a non-standard cache toggle. The 2018 prototype thus only partially meets the execution qualities. Learnability was addressed together with the evolution requirements, but responsiveness was singled out because of its importance. Responsiveness alone was treated as the measurement for system performance.

3.3 Simplification of the Code Base

Redundant code, which is not in use and not part of tests, as well as code which was deemed unnecessarily complex was simplified or extracted and removed from the prototype. Simplification improves extensibility since it leaves less code to debug and it becomes easier to read and alter what is left. The cache toggle discussed in Chapter 2.6.3 was also removed, improving both extensibility by simplifying the code, and learnability by adhering to standard web browser design.

3.4 Making Iterative Performance Improvements

As previously mentioned, performance was here defined as responsiveness and measured as time to first content and loading time. In an iterative process, the performance bottleneck was identified through testing and then the development efforts were concentrated on improving that specific aspect of the system. When the improvement was great enough fort the bottleneck to shift to another part of the system, development efforts shifted accordingly.

The initial bottleneck had been previously identified as the content extraction, due to the jsdom package utilised in this operation [3]. In order to solve this issue, the package had to be replaced. This issue was tackled first (see Chapter 4.3.1).

After significant performance improvements were achieved, the bottleneck shifted to the compression process. The algorithm either had to be optimised in regards to run-time, or the experienced delay had to be decreased by other means, such as interleaving the sending of data with the compression process. Since no other algorithm which would meet the needs of the system, as well as perform faster than the current one, was found, the second method of optimisation was used. Improving on the delay caused by compression was addressed accordingly (see Chapter 4.3.2).

3.5 UI and its Role in Evolution Qualities

Keeping the user, and the UI in mind from the start generally eases development since it acts as guide as work progresses. The UI influences scalability, extensibility, and learnability. One prime example of UI's influence on extensibility is the possibility of implementing tabs in the future, something that would aid learnability since it conforms to the design of most modern web browsers. In order to ease future development of such a feature, the underlying design needs to enable duplication or easy adaption of the web browser feature, as well as the storage and access to browsing history and cached pages.

In order to gain awareness of the factors and elements which can influence the software design, a few user tests were conducted. Some heuristic inspection can be made by the development team, based on commonly accepted design principles, but this runs the risk of biased analysis [19], especially since the development team is not specialised in UI design. The target audience for the system is a variety of people from different backgrounds, meaning the UI design needs to accommodate a broad range of users. Most notably, there will be non-expert users which may not be comfortable performing complex operations. The development team cannot hope to approximate the behaviours and needs of these users.

Since, as mentioned, UI design is not a focus of the project but rather an element of exploration in favour of the system design, the scope of user testing was severely limited. Only two full-length tests were conducted, one with an expert user and one with a non-expert user. The expert user is someone very comfortable with computers who can utilise advanced functionality in regular web browsers. The non-expert user is someone less comfortable with computers who can be expected to only utilise basic functionality in standard web browsers.

Both tests were conducted on the 2018 prototype as part of fact-finding. The tests were conducted as direct observations with a passive observer from the research group, followed by flexible interviews in which leading questions were avoided [19].

During the tests, minimal prompts were given in order to obtain less biased tests. The users were asked to search the web for something of their own choosing and to talk out loud about what they were thinking and experiencing.

The test participants were not part of the development team and did not have any background knowledge about the application or the implementation details. The application was presented as a browser for text based browsing via satellite. The tests were conducted using a regular internet connection since they focused on the users' interaction with the application and any complications from the satellite connection could be disregarded. The main difference ought to be that loading times are longer using the Iridium GO! device. The loading time is an important factor which was tackled separately. The test results were anonymised to protect the privacy of the participants.

The resulting data was a list log of the users' interactions with the system, including notes on the observer's understanding of the situation, followed by a short text describing the main points of the post-session interview. This type of logs are simple and lack a lot of detail, but are quick to write and thus fit the limited scope. The data was analysed to explore UI issues and mitigate the effect of biases held by the research team.

3.6 Documentation Efforts for the Future

A complicating factor for the development of a 2019 prototype was the distinct lack of technical documentation. The 2018 thesis contains much useful information about the design choices and the reasoning behind these. The custom protocols are also described in great detail. However, the specific implementation of the client and the server is not thoroughly documented. This lack of documentation complicated the reading of code, fact-finding, and initial testing, which led to a lengthy preparation phase.

During the project, a conscious effort was made to document the code in three levels of granularity. Line by line documentation was added in parts of the code though a more extensive use of comments. Component level documentation was added through readme files written in Markdown which offer guidance to developers exploring the 2018 code. These readme files contain short explanations of the content available in the different source files and related folders. An overview of the entire system and code base, a technical documentation, was also written and added to the top level of the file hierarchy. The documentation contains information on the file structure, how it relates to the components of the system, as well as relevant

implementation details and known issues.

This layered documentation should ease future development of the system by facilitating understanding of the existing prototype and easing entry for a future research group. Having this documentation can help cut down on preparation time as well as avoid duplication of work.

4

The 2019 Prototype

By applying the described development processes and considerations to the 2018 prototype, a new 2019 prototype was developed which more closely conforms to the requirements specification given in Chapter 3.2. The 2019 prototype detailed here is faster, more robust, and offers more aid to future developers seeking to refine the web browsing solution.

4.1 UI Analysis and System Design Considerations

As detailed in Chapter 3.5, two full-length user tests were conducted on the 2018 prototype and the results were analysed to aid in designing the 2019 prototype. The main findings and any actions taken based on the tests are presented here. Complete test logs can be found in Appendix A.

Disregarding problems which occurred due to know bugs, there are five main issues in the 2018 UI. Firstly, the conducted user tests confirmed that the cached pages toggle is unintuitive to some users. The toggle was removed during development of the 2019 prototype both because of its unintuitive nature and for data saving purposes (see Chapter 4.3.3).

Secondly, when inputting an erroneous website address, the 2018 application displays a message stating the request is being sent when in actuality the server has already received the request and gotten an error. The expert user waited for several minutes before reloading, which caused the same issue to repeat. The user never understood what was happening during the test. This indicates that input errors are not properly communicated to the user. Changes were made so that the most

common errors are communicated to the client in the 2019 prototype (see Chapter 4.4).

Thirdly, the users expected search fields on pages like Google and Wikipedia. These sites are not designed for browsing solely via links, but expect an input. Since forms and inputs are not part of the prototype functionality, navigation on these pages becomes significantly more difficult. It took a moment for the expert user to realise there was no search field, and to figure out how to access a specific Wikipedia page despite this hindrance, by using the Google search functionality from the address bar of the browser.

Similarly, the vertical menu displayed at the top of the page led to confusion. The non-expert user did not instantly recognise the navigation menu of a website as such. Users have a mental model of a navigation menu and may thus expect it to he horizontally aligned along the top, rendering the vertical display counter-intuitive [17]. While the possibility was briefly explored, neither search fields nor a horizontal menu were implemented in the 2019 prototype due to time and scope constraints.

Finally, the expert user was surprised to learn data is being sent in the background by the 2018 prototype when a cached page is displayed on screen. In most web browsers this lack of transparency would not be an issue, but since data can be costly on a satellite connection, the user needs more control. The Internet Society also recommends that an end user should be aware of the use of a PEP when one is employed [7]. However, a real user could be expected to know this after making the choice to install and use the software. The automatic sending of data was disabled during the development of the 2019 prototype (see Chapter 4.3.3).

4.2 Disabling of the Custom Encryption

For simplification purposes, the 2018 custom encryption was disabled since it is not very effective and does not add any significant functionality to the product at this stage of development. Any bugs which may appear during development were thus not relevant to find or solve and the code could be disregarded. During testing, the encryption of the 2018 prototype was disabled as well because it appeared to cause malformed packet errors.

There is also an encryption available on the Iridium Satellite Network itself. The Iridium SIM follows the Global System for Mobile Communications specifications, providing A3 and A5 which are used for authentication of the SIM card and generation of a cipher key respectively [20]. However, since the server is not directly

connected to an Iridium ground station, this encryption is not valid end-to-end.

4.3 Performance Improvements

In the 2018 prototype, there is a significant delay between requesting a website and delivery of the first content which can be displayed to the user. The requested website has to be downloaded by the server, then processed in whole, going through each stage, data extraction, link extraction, rendering, and compression, before any data can be sent to the client. Especially large pages experience significant delays. The efforts made to improve performance, as previously defined in terms of time to first content and loading time, are detailed here.

4.3.1 Circumventing the Document Object Model

A previously known bottleneck of the 2018 prototype is the jsdom package used for content extraction [3]. Especially large pages cause long delays, and long pages with a lot of links are the greatest issue. This has been addressed by replacing jsdom with the parse5 package.

The jsdom package creates a document object model (DOM), which represents the HTML file [21]. The DOM can then be searched and manipulated, but it also contains a lot of automatically created meta data. The parse5 package creates a tree of Node objects and contains a lot less information [22]. Some of the functionality which jsdom provides had to be implemented for use with parse5. jsdom provides a search function for HTML tags in the DOM-tree as well as functions for removing tags. These functions had to be implemented anew on the general tree structure provided by parse5. Traversing the parse5 tree takes less time than utilising jsdom, as evident in the following tests.

In Table 4.1, the execution time for the data extraction when requesting four different web pages is listed for the implementations using jsdom and parse5 respectively. As detailed in Chapter 3.1, both versions of the code were run on the same machine. The extraction is significantly faster when using parse5 for large websites such as Wikipedia and Aftonbladet. The number of links on each web page has the greatest impact on the run-time and is thus given in the table.

URL	jsdom (ms)	parse5 (ms)	num. of links
en.wikipedia.org/wiki/Sweden	14426	682	2416
www.aftonbladet.se	2315	290	148
www.iridium.com	830	775	127
example.com	238	232	1

Table 4.1: Execution time for data extraction in implementations utilising jsdom and parse5. Both tests were performed using a regular internet connection.

4.3.2 Implementing Faster Compression

As mentioned in Chapter 3.4, the time to first content bottleneck shifted to the compression once the jsdom package had been replaced with the parse5 package as described above. One way to improve time to first content is to send content sooner. In order to achieve this in the 2019 prototype, compression is executed on smaller sections of content, and said content is then sent before compression of the entire web page is complete. This method increases the total time it takes to compress a web page due to the overhead of calling the function multiple times. However, the first content arrives sooner, and as long as the compression of the remaining content does not delay the sending of packets, the increase in total compression time should not be an issue. Since packets can only be sent as slowly as the low bandwidth can transport them, interleaving compression and sending results in a concurrent process.

To implement these changes, the custom protocols introduced in Chapter 2.4 had to be adapted. The 2018 protocols designed to send one web page at a time, without interruptions. The data is first compressed and then sent, which means information about the total amount of data transmitted is available when the first packet is sent. With interleaving processes, the length of the compressed page is unknown when the first packet is sent. Therefore, a new end of transmission message was implemented. Furthermore, each data packet needs to be identifiable as part of the current request in order to ensure no pages become tangled upon receiving if a request is cancelled or otherwise interrupted before the end of transmission message is received.

Unfortunately, the request ID adds to the size of the packet header, but without it the changes could not be implemented. The presence of a request ID in each packet may also ease implementation of tabs, since packets belonging to different sites can be sorted from another.

The Control Channel Protocol (CCH) is a custom protocol required to request web pages, re-transmit content, enable encryption and abort transmissions. When re-

a: Original Data Transport Protocol format from 2018 [3].

bit	0	1	2	3	4	5	6	7
0 8	sequence number							
16	seq. num.				checksum			
24 ... n-24	payload							
n-16 n-8	CRC16							

b: Altered Data Transport Protocol format from 2019.

bit	0	1	2	3	4	5	6	7
0 8	sequence number							
16	seq. num.				checksum			
24 32	request ID							
40 ... n-24	payload							
n-16 n-8	CRC16							

Table 4.2: The original and altered version of the Data Transport Protocol. The table shows what kind of information is encoded in each bit. For example, the first row shows that bit 0 to bit 19 contain the sequence number, which identifies the order of the packets.

questing a web page, it is either requested via URL or via link ID, as detailed in Chapters 2.6.1 and 2.3. Upon making a URL request, a RequestURL response is returned. Upon making a link ID request, a RequestLink response is returned. The original and altered version of these responses as shown side by side in Tables 4.3 and 4.4.

The original 2018 RequestURL response gives the content length of the requested web page as well as the first and last sequence numbers of related data packets in order to identify the first and last packet belonging to the web page. The compression parameter defines which dictionary to use for decompression (see Chapter 2.5). Similarly, the original RequestLink response gives this information, as well as the URL corresponding to the requested link ID, so it may be displayed to the user.

The 2019 protocols provide the request ID instead of the last sequence number. Having both request ID and last sequence number would bloat the header of the packets. As is, the 2019 header is 8 bits smaller because the request ID is shorter than a sequence number.

The absence of a last sequence number means the last data package can no longer be identified using these RequestURL and RequestLink response messages. In the 2019 prototype, the end is instead marked by a new End of Transmission packet in the control channel, as seen in Table 4.5. The End of Transmission message contains the request ID and the last sequence number. With this message, the last data package can be identified for the specified request ID.

a: Original RequestURL response from 2018 [3].

bit	0	1	2	3	4	5	6	7
0	3							
8 16 24 32	content length							
40 48 56	first seq. num.							
64 72 80	last seq. num.							
88	compression parameter							

b: Altered RequestURL response in 2019.

bit	0	1	2	3	4	5	6	7
0	3							
8 16 24 32	content length							
40 48	request ID							
56 64 72	first seq. num.							
80	compression parameter							

Table 4.3: The original and altered version of the RequestURL response. The table shows what kind of information is encoded in each bit. For example, the first row shows that bit 0 to bit 7 contain the number 3, which identifies the message as a RequestURL response.

a: Original RequestLink response from 2018 [3].

bit	0	1	2	3	4	5	6	7
0	5							
8 16 24 32	content lenght							
40 48 56	first seq. num.							
64 72 80	last seq. num.							
88	Compression parameter							
96 ...	URL							

b: Altered RequestLink response in 2019.

bit	0	1	2	3	4	5	6	7
0	5							
8 16 24 32	content length							
40 48	request ID							
56 64 72	first seq. num.							
80	Compression parameter							
88 ...	URL							

Table 4.4: The original and altered version of the RequestLink response. The table shows what kind of information is encoded in each bit. For example, the first row shows that bit 0 to bit 7 contain the number 5, which identifies the message as a RequestLink response.

bit	0	1	2	3	4	5	6	7
0	8							
8 16	request ID							
24 32 40	last seq. num.							

Table 4.5: End of Transmission message which gives the last sequence number for a specific request ID, identifying the last data packet of the relevant request. This message type is new for the 2019 prototype. The table shows what kind of information is encoded in each bit. For example, the first row shows that bit 0 to bit 7 contain the number 8, which identifies the package as an End of Transmission message.

While the 2018 prototype compresses the entirety of the text with both dictionaries before deciding which version to send, the 2019 prototype only compresses the first 2048 bytes with both dictionaries. After the initial double compression, it is assumed that the dictionary which produced the most efficient compression will continue to be most efficient, and is used to compress the rest of the packages. In essence, the language of the web page is guessed based on which compression was most efficient.

An alternative to this method would be to check the HTML document of the website and search for a language tag. Since not all web pages have these, a secondary method to pick the dictionary would be required either way. Implementing such a solution would also require a parser to seek out this information before extraction, not to mention compression. Since such a solution complicates the code and does not reliably solve the problem of picking a dictionary, it was not implemented.

4.3.3 Quick Access Browsing History and Cached Pages

As detailed in Chapters 2.6.2 and 2.6.3, browsing history functionality as well as caching of visited websites is available in the 2018 prototype, and saved in separate files. Furthermore, a request is sent in parallel to displaying a cached page in order to provide the user with an updated version as soon as possible. In the 2019 prototype, cost management is prioritised and this non-essential background process was removed to save data. If the user wishes to reload the page, they can do so via the reload button in the navigation bar. The GUI was altered accordingly, removing the cache indicator from the status bar.

In the 2019 prototype, both history and cache implementations have been replaced with internal data structures. Since adding to the browsing history and cache is

expected to happen often, it is quicker to save it in an internal data structure rather than utilising a file handler. If saving of this data between sessions is desirable, it could be saved to a file once at the end of the session. Most importantly, the new implementation is simpler, better isolated from the rest of the code, and easy to duplicate, aiding extensibility and scalability.

No working ready to use implemented of a doubly linked list could be found and successfully added to the prototype during development. The data structure utilised in saving the browsing history in the 2019 prototype is an array and a number which acts as iterator. Using these two variables, the same functionality as a doubly linked list is achieved without much custom implementation. The solution makes access to the previous and next site a O(1) operation. Saving a new web page to history (the most common use case) is an O(n) operation, but in most cases it will be O(1) since the new page is added after the one the iterator is pointing to at the time of insertion, meaning it is a simple append unless the iterator is not at the last position. On occasion the operation will cause a copy operation of the history array with complexity O(n).

The cache implementation was planned as a hash map which allows for quick access to web pages based on their ID. Using a map enables the browser to retrieve a cached version of a web page independent of where in the history it appears, or if it appears in the history at all. Tabs can share the same cache. If so desired, the cache could be saved to a file once at the end of a session and used again during succeeding sessions. A time stamp should inform the user of the age of the cached page and the user can always request an updated version by clicking reload. The cached version of the page should then be replaced by the more recent version, and the time stamp updated accordingly. Similarly to the history implementation, this cache implementation would be simpler, better isolated, and easy to duplicate, aiding both extensibility and scalability. Due to time constraints, this cache implementation was not completed.

4.4 Handling Communication Errors

In the 2018 prototype, errors that occur on the server are not communicated to the client. This lack of communication causes issues where the client waits for an answer to its request despite the server no longer working on said request. In some cases, the server can crash and the client is not notified. In the 2019 prototype these errors are caught and handled by sending a data package containing an error message instead of web content, which is then displayed to the user.

Sending error messages for the user to read is a great improvement since it lets the user know what is happening server side and makes the system more robust. However, it would aid extensibility if error messages were easily identifiable by the client application. The same result could be achieved by extending the custom protocols to include an error message type which contains error codes the client application can process as it wishes. This would help isolate the client and server from another which eases altering or replacing components. The message can still be directly displayed to the user if the client application designer wishes it. This solution was not implemented due to time constraints.

5

How Well the 2019 Prototype Meets the Requirements

The functional requirements met by the 2018 prototype are also met by the 2019 prototype. Improvements regarding functional requirements are limited to bug fixes and error handling as described in Chapter 4.4. Evolution qualities and learnability were considered in the design of the 2019 prototype as detailed in Chapter 4, but were not measured. The quantitative tests were limited to two performance markers, time to first content and loading time.

As described in Chapter 3.1, the 2018 and 2019 prototypes were run on the same machine to enable comparison of their performance. Two types of tests were conducted, field tests with the Iridium GO! device, and performance tests without the device. The connection using the Iridium GO! device is very sensitive to access to open sky, such as on the sea. Due to this limitation, maintaining a stable connection in the city is complicated and only a few field tests could be conducted. Complete field test results are presented here along with examples from the performance tests. Extensive performance test data can be found in Appendix B.

Field tests with the Iridium GO! device were run using a separate client and server to get a close approximation of conditions in the field. The server was run from a digital server hosted by DigitalOcean in London. The Iridium GO! device and client were used from Gothenburg, Sweden. No tests in remote locations were conducted. In theory, this should not make much difference since the Iridium Satellite Network is globally available, but this could not be confirmed. Performance tests were run using the same set-up, but using a regular internet connection instead of the Iridium Satellite Network between server and client.

5.1 Time To First Content

Time to first content was measured from the time of request until the first packet arrives to the client. As stated in the requirements specification, time to first content shall not exceed 10 seconds and should be as short as possible.

URL	2018 (ms)	2019 (ms)
bbc.com	1892	605
dn.se	2149	842
sv.wikipedia.org/wiki/Sjökort	1294	662
en.wikipedia.org/wiki/Sweden	16080	958
Sample mean time to first content	**2205.28**	**756.92**

Table 5.1: Examples of time to first content in the 2018 and 2019 prototype without Iridium GO!. The mean of the complete sample is presented in the last row. The complete table can be found int Appendix B

In Table 5.1, a few examples from the performance test without Iridium GO! are presented. The time to first content is well below 10 seconds for small web pages using the 2018 prototype and for almost all pages using the 2019 prototype. The time to first content is notably shorter in the 2019 prototype, with a reduction of 1448.36 ms on average compared to the 2018 prototype.

URL	2018 (ms)	2019 (ms)
en.wikipedia.org/wiki/Law_of_the _sea	4658	11591
en.wikipedia.org/wiki/United_Nations _Convention_on_the_Law _of_the_Sea	3091	5407
smhi.se	25981	8969
koket.se	8337	5108
www.livescore.com	2996	6485
Sample mean time to first content	**9012.6**	**7512**

Table 5.2: Time to first content in the 2018 and 2019 prototype, using Iridium GO!.

When using the Iridium GO! device, delays are less predictable, as demonstrated in Table 5.2. It appears from the sample that the average time to first content has decreased by 1500,6 ms on average for the 2019 prototype compared to the 2018 prototype. This comes close to the decrease without the Iridium GO! device. However, due to the small sample size, it cannot be stated with any certainty that

this is indeed the mean of the time to first content when using Iridium GO!. Despite this uncertainty, the results strengthen the previous claim of an approximately 1.45 seconds improvement.

5.2 Loading Time

Loading time is defined as follows.

$$\frac{complete\ load - time\ to\ first\ content}{page\ size}$$

Time to first content is the same as previously defined. Complete load is the time from request until the last package arrives at the client.

URL	**size (byte)**	**2018 (ms/byte)**	**2019 (ms/byte)**
bbc.com	12111	2.557839	2.608951
dn.se	22650	2.495011	2.557528
sv.wikipedia.org/wiki/Sjökort	13513	2.365130	2.341523
en.wikipedia.org/wiki/Sweden	215428	2.502168	2.526157
Sample mean loading time	-	**1.725189**	**1.800451**

Table 5.3: Examples of loading times in the 2018 and 2019 prototype without Iridium GO!. The mean of the complete sample is presented in the last row. The complete table can be found int Appendix B

In Table 5.3, a few examples from the performance test without Iridium GO! are presented. The average loading time of the 2019 prototype is slightly longer than that of the 2018 prototype. As detailed in Chapter 4.3.2, the run-time of the compression is longer than before because it runs in smaller chunks and is interleaved with the sending of data. The improvement to time to first content is much more significant, however, making the 0.075 ms/byte increase in average loading time an acceptable performance trade-off.

When using Iridium GO!, the delays are as previously stated less predictable. This affects the loading time, as presented in Table 5.4. The average loading time appears to have increased for the 2019 prototype compared to the 2018 prototype. The result strengthens the claim that there is an increase in loading time, but the magnitude

URL	size (byte)	2018 (ms/byte)	2019 (ms/byte)
en.wikipedia.org/wiki/Law_of_the_sea	4449	2.452910	2.939987
en.wikipedia.org/wiki/United_Nations_Convention_on_the_Law_of_the_Sea	25432	3.136796	4.092207
smhi.se	28116	3.147247	4.35567
koket.se	15290/14452*	14.4743**	3.345520
www.livescore.com	8994	2.906160	3.980543
Sample mean loading time	-	**2.910778**	**3.742785**

Table 5.4: Loading Time in the 2018 and 2019 prototype, using Iridium GO!.
* = Page size varied between tests. Displayed as: version 2018/2019.
** = Packages were lost and had to be reloaded.

of the change is unclear. Again, no definitive conclusion can be drawn due to the small sample size. When not using Iridium GO!, the loading time increases with 0.075 ms/byte on average, but whether the increase when using Iridium GO! is approximately the same or significantly higher (as it appears in the sample) cannot be determined with any certainty. In either case, even assuming that the increase is in fact 0.832 ms/byte on average, the decrease in time to first content is still more significant.

5.3 Suggestions for Future Work

While the 2019 prototype meets the specified requirements better than the 2018 prototype, there is room for improvement. The new technical documentation available for the prototype should ease future development efforts. The main suggestions for areas of improvement are also presented here.

Firstly, if one server is supposed to run for a multitude of clients, scalability needs to be addressed. As detailed in 2.3, the number of links the server needs to keep can quickly escalate. One solution would be to request links by both ID and the URL from which the link was clicked. The server can then safely discard old link ID and URL matches and find them anew by checking the URL from which the link request was made, recreating the link ID and URL match, when necessary. The exception would be highly dynamic pages such as news sites where the links change over time.

Saving data is of high priority. Less information could be sent if only the content currently visible on screen is sent to the client rather than entire web pages. If the user does not choose the scroll down, no more than a few lines past what is currently visible on screen are needed to provide the same user experience. This solution requires a fast enough system that the loading times which occur once the user chooses to scroll down do not frustrate the user. Where the critical line for this delay is would require further study.

Another way to save data is to implement a digital assistant, such as Google Assistant, which can answer questions the user wishes to research in a minimal format. Providing the currently available Google Assistant was not feasible during the project since the functionality is severely reduced when only text based content is returned. Many of the most common uses for the assistant, such as setting timers or requesting images, become unavailable. When not using such a pre-filtered version of the assistant, it often returns information in the form of pictures or heavily stylised boxes which the PEP filters out, leaving only an empty page to send to the client.

An alternative solution would be to implement a simple on-server search which only returns the blocks of text containing given search terms within a specified web page.

Safe browsing is also important. As detailed in Chapters 2.4 and 4.2, the encryption of the 2018 prototype is not very effective and the encryption provided by the Iridium SIM does not cover the connection end-to-end. Any web browsing service ought to consider the necessity of reliable end-to-end encryption. Especially if data is to be sent from the client to the server in the future to enable functionality like logging in to forums or filling in forms, the encryption needs to be investigated further. In this event, the known security flaw of the 2018 encryption, which leaves it vulnerable to active attacks, needs to be taken into account. It may also be prudent to warn the user against logging in to sensitive pages and refrain from online banking if this issue is not resolved.

Finally, there are several issues with the current UI, as detailed in Chapter 4.1. The main issues are missing functionality and unusual presentation of information. In addition, many common hotkeys (such as Alt+Left Arrow for history navigation) do not work in the current prototype, which may cause irritation due to habituation [17]. The 2019 prototype also provides new implementations for features from the 2018 prototype which should ease implementation of tabs, a standard web browser UI feature.

It may also be prudent to conduct a more in-depth set of user tests, including the Iridium GO! device, to gain a better understanding of how users interact with the entire system. A greater volume of, as well as more precise user stories could also

be a useful tool for future development and enable detection of design flaws or help identify inefficient task flows. Identifying primary users and their characteristics may inform future design choices as well. Neither should accessibility design be neglected, and may well cause a curb-cut effect. It is not inconceivable that adjustable text and button sizes can be beneficial to users sitting on a rocky boat or operating outdoors in cold weather.

5.4 Ethics

In the development of the web browser, a couple of ethical aspects have to be taken into consideration. A handful of ethical aspects were already pointed out in the 2018 report, such as integrity, privacy, and financial effects of the system.

During a connection, the 2018 prototype only keeps track of the client's IP-address, the current page, and the links on that page [3]. In order to enable other functionality, it might be required to store more extensive information on the proxy server in the future. It is important to ensure such data is kept secure and to prevent privacy breaches. The system needs to respect the confidentiality of the user's communication. It is important that the user feels safe using the service, and understands to which degree the service is, or is not safe.

During data extraction, it is vital to respect the integrity of the data. Whilst the amount of data sent between server and client needs to be severely limited, data filtration influences what information becomes available to the user. In the 2018 prototype, all textual information is preserved, but in order to restrict the data further a filter could be put into place to even for this information. Alternatively, if images or videos are to become available, these may be filtered in a similar manner. This kind of content filtration may even affect the user's understanding of the content on a page. In summary, any filtration of information will influence how that information is received, and this is not to be taken lightly. Users need be made aware that an automatic filtration is in place when they view the content.

There are other secondary effects to data filtration as used in the prototype. For example, monetization of web pages may be affected by filtration. In the 2018 thesis, this was summarised as follows.

> The service removes a lot of content before transmission to the client, including most advertisements, thus acting as an "ad-blocker". Many web pages rely on revenue from advertisements, and users of the prototype will not contribute to their income. However, the target audience of this

> service is small and probably visits the same pages when not in remote locations. Because of this, the negative economic impact is probably negligible [3].

Finally, when developing a web service, the larger picture of a connected world may be taken into account. There are many arguments for and against, but what is certain is that interconnectivity is a fact of the modern world and that increasing accessibility in remote areas may enable further democratisation of the available content. Internet access can be democratised not only by improving hardware, but also by using the globally available bandwidths more efficiently. Unfortunately, using satellite internet is pricey and this effect is thus greatly diminished.

During the execution of the project itself, there were few ethical aspects to take into consideration. Of note is that the user test data has been anonymized in order to respect the participants' privacy. The participants' names and personal data were not recorded. The participants were instead referred to as 'user' and categorised as 'expert' or 'non-expert', depending on their familiarity and comfort with computers. Since the tests were very limited in scope and did not include any user characterisation efforts, the loss of personal data does not affect the test results in any significant way.

6

Conclusion

The purpose of the project was stated as enabling general web browsing in remote locations in a cost-effective and timely manner, based on the 2018 prototype. Where the 2018 prototype provides a proof of concept for using a PEP to extract data in order to better utilise a low bandwidth connection, the 2019 prototype provides a functional service which can most likely be utilised by a non-expert user.

As stated in the 2018 thesis, content extraction and content streaming are most important [3], saving data and time. In order to achieve timely delivery, the time until first content and loading time are critical, as discussed in Chapter 3.1. Especially time to first content is vital and has been improved with about 1.45 seconds on average by replacing the jsdom package used for content extraction with the parse5 package, and interleaving the compression process with the sending of data as detailed in Chapters 4.3.1 and 4.3.2.

A small increase in loading time occurred due to these changes, but the performance trade-off is arguably worth it. Even if data arrives slower after the first content, the user knows the system is working and can see data arriving continuously, which amounts to a reliable and user friendly experience. Unless the page is very large, the complete page is also available to the user sooner. While the 2019 prototype is still notably slower than a regular web browser, it is significantly faster than the 2018 prototype.

With new error handling, the 2019 prototype is also more robust than the previous iteration. Not all types of media can be accessed, and not all functionality on requested websites is intact, but sites are loaded relatively quickly and the filtration of data is likely to save costs on a limited data plan. Further data, and thus costs, have been saved in the 2019 prototype by avoiding non-essential requests and reloads, as described in Chapter 4.3.3.

While time and cost are the most important factors, improvements in neither can be reliably achieved without a solid code base to work with. Evolution qualities have thus been addressed actively in the development of the 2019 prototype. Component isolation, simplicity, and scalability have been improved as detailed in Chapters 3.3 and 4.3.3. These efforts, along with new documentation, should ease future development. For such endeavours, the main suggestions for future work are described in Chapter 5.3.

In summary, the 2019 prototype, while not providing all the functionality of a regular web browser at this time, enables a reliable web browsing experience for a low bandwidth satellite connection. It is both relatively fast and saves data, which may reduce costs for the user.

The prototype was developed using the Iridium GO! device, but could be used on any internet connection, even GEO satellite networks, cellular networks, or via shortwave internet. Since it is designed with bit-error handling in mind, the solution ought to be viable for technologies which may be more prone to such errors. The work can thus be applied more generally for data extraction and transfer to remote locations.

Bibliography

[1] J. F. Kurose and K. W. Ross, *Computer networking : a top-down approach*, 7th ed. Pearson Education, 2017.

[2] Tracphone v7-hts. Accessed: 10 May 2019. [Online]. Available: https://www.kvh.com/Leisure/Marine-Systems/Mobile-Communications/mini-VSAT-Broadband/TracPhone-V7HTS.aspx

[3] T. Andersson, L. Blomkvist, A. Hast, F. Karlsson, J. Lindström, and T. Sundell, *Very Low Bandwidth (Marine) Web Surfing A Fault-Tolerant Content Streaming Web Browsing Solution*. Chalmers Tekniska Högskola, 2018.

[4] Iridium go airtime. Accessed: 14 May 2019. [Online]. Available: https://www.satphonestore.com/airtime/iridium-go-airtime-rates.html

[5] I. Communications, "Quick start guide: ios iridium mail & web app and iridium go!" [Online]. Available: http://www.groundcontrol.com/iridium/Iridium_Go_Mail_And_Web_iOS_Users_Guide.pdf

[6] Xweb. Accessed: 12 May 2019. [Online]. Available: http://www.globalmarinenet.com/product/xweb/

[7] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," Jun. 2001. [Online]. Available: https://rfc-editor.org/rfc/rfc3135.txt

[8] C. Caini, R. Firrincieli, and D. Lacamera, "Pepsal: a performance enhancing proxy designed for tcp satellite connections," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 6, 2006, pp. 2607–2611.

[9] S. Subramanian, S. Sivakumar, W. J. Phillips, and W. Robertson, "Investigating tcp performance issues in satellite networks," in *3rd Annual Communication Networks and Services Research Conference (CNSR'05)*, 2005, pp. 327–332. [Online]. Available: https://ieeexplore.ieee.org/document/1429988/

[10] GrandDeluxe. (2011) Accessed : 5 April 2019. [Online]. Available: https://commons.wikimedia.org/wiki/File:Iridium_Coverage_Animation.gif

[11] K. Maine, C. Devieux, and P. Swan, "Overview of iridium satellite network," in *Proceedings of WESCON'95*, 1995, p. 483.

[12] Follow the 8 launch missions. Accessed: 28 April 2019. [Online]. Available: https://www.iridiumnext.com/

[13] What's next? Accessed: 30 April 2019. [Online]. Available: https://web.archive.org/web/20080406131233/http://www.iridium.com/about/next.php

[14] iperf - the ultimate speed test tool for tcp, udp and sctp. Accessed: 2 May 2019. [Online]. Available: https://iperf.fr/

[15] Elinks - full-featured text www browser. Accessed: 10 April 2019. [Online]. Available: http://elinks.or.cz/

[16] D. Salomon and G. Motta, *Handbook of Data Compression*, 5th ed. New York: Springer, 2010.

[17] J. Tidwell, *Designing Interfaces*. O'Reilly Media, Inc., 2010.

[18] A. Bouch, A. Kuchinsky, and N. Bhatti, "Quality is in the eye of the beholder: Meeting users' requirements for internet quality of service," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2000, pp. 297–304. [Online]. Available: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/332040.332447

[19] D. Stone, C. Jarret, M. Woodroffe, and S. Minocha, *User Interface Design and Evaluation*. Elsevier, Inc., 2005.

[20] D. Veeneman. Iridium. Accessed: 19 March 2019. [Online]. Available:

http://www.decodesystems.com/iridium.html

[21] A javascript implementation of the whatwg dom and html standards, for use with node.js. Accessed : 27 May 2019. [Online]. Available: https://github.com/jsdom/jsdom

[22] Html parsing/serialization toolset for node.js. whatwg html living standard (aka html5)-compliant. Accessed : 27 May 2019. [Online]. Available: https://github.com/inikulin/parse5

A

User Tests

This appendix contains logs from the user tests conducted during the project. A detailed description of how these were conducted can be found in chapter 3.5.

A.1 2018 Prototype with Expert User

The test was conducted on a Windows 10 machine running both client and server and using a regular internet connection. The application was presented to the user as a text based web browser for satellite internet. The user has general computer expertise but has never seen the application before.

User action	**System response**
User enters "google" into address field	
	Application returns google search on the term "google"
User scrolls up and down the page. User asks where the search field is. User searches for search field for several seconds	
User enters "youtube" into address field	
	Application returns google search on the tern "youtube"
User scrolls down and clicks link to navigate to youtube.com	
	Application returns the text "iFrame"

Continued on next page

Continued from previous page

User action	**System response**
User seems confused. User clicks address field and presses enter	
	Application returns the text "iFrame"
User concludes the application is not working correctly	
User enters "wikipedia" into address field	
	Application returns google search on the term "wikipedia"
User scrolls up and down to find the correct link and then clicks to navigate to wikipedia.org	
	Application returns wikipedia.org
User scrolls up and down and comments on the missing search field.	
User enters "wikipedia car" into search field	
	Application returns google search on the term "wikipedia car"
User scrolls down and clicks to navigate to sv.m.wikipedia.org/wiki/Car	
	Application returns sv.m.wikipedia.org/wiki/Car
User scrolls up and down and appears confused. User realises it is a swedish page about a bug.	
User clicks back arrow	
	Application returns google search on the term "wikipedia car"
User clicks forward arrow	
	Application returns a cached version of sv.m.wikipedia.org/wiki/Car

Continued on next page

Continued from previous page

User action	**System response**
The user reads the message at the top of the screen and appears confused. The user reads the message two more times before finding the arrows in the bottom right corner and clicking it.	
	Application returns sv.m.wikipedia.org/wiki/Car
User clicks back arrow	
	Application returns google search on the term "wikipedia car"
User scrolls up and down	
User clicks forward arrow	
	Application returns a cached version of sv.m.wikipedia.org/wiki/Car
User clicks the cache pages toggle	
	Application returns sv.m.wikipedia.org/wiki/Car
User clicks address field and replaces 'sv' with 'en'	
	Application returns en.m.wikipedia.org/wiki/Car
User scrolls up and down and clicks a link.	
	Application returns en.m.wikipedia.org/wiki/Motor _vehicle
User realises they are on a mobile version of the website. User clicks address field and removed 'm.'	
	Application returns en.wikipedia.org/wiki/Motor-vehicle
User scrolls up and down, spotting no difference.	
User clicks back arrow	

Continued on next page

Continued from previous page

User action	**System response**
	Application returns en.m.wikipedia.org/wiki/Motor_vehicle
User clicks back arrow	
	Application returns en.m.wikipedia.org/wiki/Car
User realises they are on a mobile version of the website. User clicks address field and removes 'm.'	
	Application returns en.wikipedia.org/wiki/Car
User scrolls up and down, spotting no difference.	
User clicks help/learn more	
	Standard browser on the machine starts and loads electronjs.org
User is surprised and returns to application. User clicks help/documentation	
	Standard browser on the machine starts and loads the documentation page github.com/electron/electron/tree/master/docs#readme
User sighs and returns to application. User clicks view/toggle full screen	
	Application is displayed in full screen
User clicks view/toggle full screen	
	Application is displayed in window
User clicks file/open. User clicks two more times. User clicks file/close	
	Application stops.
User appears confused. User starts application again.	

Continued on next page

Continued from previous page

User action	**System response**
User is prompted to go to example.com User enters "example.com" into address field	
	Application returns example.com
User is prompted to go to example.se	
User enters "example.se" into address field	
	Application shows message "sending request"
User waits for over two minutes. User clicks address field and presses enter.	
	Application shows message "sending request"
User waits for nearly 30 seconds. User clicks address field and presses enter.	
User concludes the application is not working.	

During a flexible interview following the test, it was revealed that example.se is not a viable page. The user was surprised and would have expected an error message. The user was asked what happened when a cached page was shown. The user concluded it was a cached page and was surprised to learn information was being sent in the background. The user suggested search fields on google.com and wikipeida.org would be desirable.

A.2 2018 Prototype with Non-Expert User

The test was conducted on a Windows 10 machine running both client and server and using a regular internet connection. The application was presented to the user as a text based web browser for satellite internet. The user has never seen the application before.

User action	**System response**
User enters "google" into address field	
	Returns google search on the term "google"
Clicks “Images” link	
	Loads the ’Images’ page on the search term "Google"
Clicks on URL “Google”	
	Returns user to google start-page
Clicks on the link “Log in”	
	Prompts user that the page is loading. Prompts briefly that the user is disconnected. Arrives at the google login page.
Tries to figure out where to login. Gives up after a while. Enters “www.gp.se” into address field.	
	Returns the address “http://www.gp.se”
Tries to click the cut of part of a link Thinks the new page is loading because it says “Loading” in the bottom left corner when if fact the user missed the button, and it is the current page that is still loading. User tries to figure out if it opened in a new page or tab. Waits but nothing happens. Instead clicks “Sport -> Football”	
	Returns address “http://gp.se/sport/football”

Continued on next page

Continued from previous page

User action	**System response**
User first thinks they are on the same page since the first content is the same. (Information about cookies and menus.) Scrolls down and finds the content. Scrolls up again and once again tries the cut of link, but clicks correctly this time.	
	Returns address “http://gp.se/nyheter/göteborg”
Scroll down, finds the article and clicks on one	
	Loads an article
Understand now that the user have to Scroll down to get to the content. Reads the article. Enter ”www.smhi.se” into the address field	
	Loads the page
Reads about cookies in confusion Realise checking the weather might not work since it usually consists of images. Finds the link ”10-dygnsprognos” and clicks on it.	
	Return the page
Scrolls around to find out about the weather but can’t find anything. Finds the link ”10-dygnsprognos” again and clicks on it. In frustration, the user states that there is a lot of content but nothing about the actual weather. Enters “www.eniro.se” into the address field.	
	Returns page
Clicks the link “Vem Ringde?”	Returns page.
Looks around for a search field to enter a phone-number, but cannot find anything.	
Clicks the link “Privatpersonner”	

Continued on next page

Continued from previous page

User action	**System response**
	Returns the page
Keeps looking for a search field but can't find anything.	
Clicks the link again "Vem Ringde?"	
	Returns page.
Keeps Looking for a search field. Gives up after a while.	
Enters "svt.se" into adress field, testing if "www" is required or not.	
	Returns "svt.se"
Click the link "TV-Tablå"	
	Returns the page
Thinks it's not working util the user Scrolls down and finds the desired content Clicks "Start" link	
	Returns the page
Attempts to click on the link Trädgårdstider, but misses the button. Thinks the page is loading, when if fact the current page is still loading. In slight confusion, clicks the link "Trädgårdstider avsnitt 5"	
	Returns the page.
User states that this is pretty much what the infancy of the internet was like, slow, and this type of content.	
Enters "korstörne" into address field	
	Returns google search of "korstörne"
Waits for the page to finish loading. Scrolls down and states that it is possible to go to the google articles.	
Clicks link ""korstörne blommor"	
	Returns google search of "korstörne blommor".

Continued on next page

Continued from previous page

User action	**System response**
Realise that user won't find any pictures.	
Enters "Chilli frö groning" into address field.	
	Returns google search of "Chilli frö groning".
Scrolls down and clicks the first link.	
	Returns www.olda-chilli.se

After the testing was done the user was asked the broad question on what the experience was like. The user responded that it was unclear how to preform a search query on pages like 'Eniro'. The user also found it very confusing that you had to scroll down quite far to find the actual content.

B

Performance Test Results

This appendix contains the performance test data without Iridium GO! referenced in chapter 5.

B.1 Time to First Content

Table B.1: ttfc = Time To First Content

URL	**2018 ttfc (ms)**	**2019 ttfc (ms)**
bbc.com	1892	605
dn.se	2149	842
svd.se	5128	2298
www.hlr.nu/sa-har-gor-du-vuxen-hlr	2321	1186
www.hlr.nu/forsta-hjalpen	2624	1115
vandringsguiden.se/planera-vandringen/kompass-och-kartlasning	3193	2310
sv.wikipedia.org/wiki/Sjökort	1294	662
www.jordbruksverket.se/amnesomraden/landsbygdfiske/branscherochforetagande/fritidsfiskeochfisketurism/fritidsfiske.4.e01569712f24e2ca0980009630.html	1557	720
www.havochvatten.se/hav/fiske–fritid/sport–och-fritidsfiske/fiskeregler/fiskeregler-for-fritidsfiske.html	1892	729
www.britannica.com/topic/fishing-recreation	2361	1109

Continued on next page

Continued from previous page

URL	**2018 ttfc (ms)**	**2019 ttfc (ms)**
en.wikipedia.org/wiki/Iridium_satellite_constellation	1961	597
www.un.org/en/sections/issues-depth/oceans-and-law-sea/index.html	2578	1109
en.wikipedia.org/wiki/Law_of_the_sea	1335	1012
en.wikipedia.org/wiki/United_Nations_Convention_on_the_Law_of_the_Sea	1673	619
smhi.se	2765	1061
koket.se	2605	1527
www.livescore.com	1724	464
en.wikipedia.org/wiki/Sweden	16080	958
Sample mean time to first content	**2205.28**	**756.92**

B.2 Loading Time

Loading time is defined as follows.

$$\frac{completeload - timetofirstcontent}{pagesize}$$

Table B.2: lt = Loading Time. If page size varies between tests, it is given anew for 2019

URL	**size (byte)**	**2018 lt (ms/ byte)**	**size (byte)**	**2019 lt (ms/ byte)**
bbc.com	12111	2.557839		2.608951
dn.se	22650	2.495011		2.557528
svd.se	17680	2.535351	17973	2.581483
www.hlr.nu/sa-har-gor-du-vuxen-hlr	4847	2.224675		2.392614
www.hlr.nu/forsta-hjalpen	6286	2.224944		2.349666

Continued on next page

Continued from previous page

URL	**size (byte)**	**2018 lt (ms/ byte)**	**size (byte)**	**2019 lt (ms/ byte)**
vandringsguiden.se/planera-vandringen/kompass-och-kartlasning	5500	1.963636		2.302364
sv.wikipedia.org/wiki/Sjökort	13513	2.365130		2.341523
www.jordbruksverket.se/amnesomraden/landsbygdfiske/branscherochforetagande/fritidsfiskeochfisketurism/fritidsfiske.4.e01569712f24e2a0980009630.html	9533	2.250498		2.435750
www.havochvatten.se/hav/fiske–fritid/sport–och-fritidsfiske/fiskeregler/fiskeregler-for-fritidsfiske.html	12689	2.446923		2.573646
www.britannica.com/topic/fishing-recreation	22415	2.333125		2.400535
en.wikipedia.org/wiki/Iridium_satellite_constellation	39093	2.401479	39789	2.491091
www.un.org/en/sections/issues-depth/oceans-and-law-sea/index.html	12042	2.395532		2.452084
en.wikipedia.org/wiki/Law_of_the_sea	4449	2.182513		2.371994
en.wikipedia.org/wiki/United_Nations_Convention_on_the_Law_of_the_Sea	25432	2.266790		2.32341
smhi.se	28116	2.466958	28256	2.500708
koket.se	14452	2.659148	15190	2.636932
www.livescore.com	8994	2.858016		2.930620
en.wikipedia.org/wiki/Sweden	215428	2.502168		2.526157
Sample mean loading time	-	**1.725189**	-	**1.800451**